Policy-Governed Self-Healing Loops for Observability Pipelines in Distributed Cloud Systems

Sneha Gullapalli
Independent Researcher
Austin, TX, USA

ABSTRACT

Observability pipelines have become a critical component of distributed cloud systems, collecting and processing metrics, logs, traces, and events that support monitoring, diagnosis, and automated operations. Failures within these pipelines, including queue saturation, exporter outages, schema drift, timestamp delays, memory pressure, and backend unavailability, can significantly reduce system visibility and affect operational decision-making. This paper presents SHIELD-OP, a policy-governed self-healing framework designed to improve the resilience of observability pipelines. The framework extends the Monitor–Analyze–Plan–Execute (MAPE) loop by incorporating policy constraints, verification mechanisms, rollback procedures, cooldown controls, and risk-aware remediation actions. A reproducible simulation study involving 50 random seeds and 120 failure episodes per seed compares SHIELD-OP with static alerting, retry-based recovery, ungoverned automation, and MAPE-K without verification. Results indicate that SHIELD-OP reduces recovery time and telemetry loss while maintaining a lower rate of unsafe actions, demonstrating the value of governance-aware self-healing for cloud observability infrastructure.

General Terms

Reliability, Design, Verification

Keywords

Self-healing observability, AIOps, OpenTelemetry, policy governance, MAPE-K, telemetry pipelines, automated remediation, distributed cloud systems

1. INTRODUCTION

Modern cloud-native systems rely heavily on observability pipelines to collect, process, and deliver telemetry data such as metrics, logs, traces, and events. These pipelines are commonly viewed as supporting infrastructure, yet they constitute complex distributed systems with their own queues, exporters, storage backends, resource constraints, and operational dependencies. When failures occur within the observability pipeline, the quality and availability of telemetry data can be significantly affected, reducing the effectiveness of monitoring, diagnosis, and automated operational decision-making.

The growing adoption of AI-assisted operations further increases the importance of reliable telemetry. Techniques such as anomaly detection, root-cause analysis, incident summarization, and automated remediation depend on timely and accurate observability data. However, failures including queue saturation, exporter outages, schema drift, timestamp delays, memory exhaustion, and backend unavailability can compromise the telemetry stream and limit the effectiveness of these operational capabilities. While simple retry mechanisms may resolve transient failures, they can also increase resource pressure under sustained fault conditions. Similarly, fully automated recovery actions may introduce additional risks if remediation decisions are executed without adequate validation.

To address these challenges, this paper presents SHIELD-OP, a policy-governed self-healing framework for observability pipelines in distributed cloud environments. Unlike traditional self-healing approaches that focus primarily on application services, SHIELD-OP targets the observability infrastructure itself. The framework integrates continuous monitoring, fault classification, policy-aware action selection, remediation execution, verification procedures, rollback mechanisms, and human review for high-risk situations.

This study investigates four research questions: (RQ1) whether policy-governed remediation can reduce recovery time while maintaining operational safety; (RQ2) whether verification and rollback mechanisms can reduce telemetry loss following recovery actions; (RQ3) whether high-risk failures benefit from human review instead of full automation; and (RQ4) whether OpenTelemetry-Collector-inspired stress scenarios reveal additional operational risks beyond those observed in generic simulation environments.

The main contributions of this paper are as follows: (1) a policy-governed self-healing architecture for observability pipelines in distributed cloud systems; (2) a fault classification and action-safety framework for remediation decision-making; (3) a reproducible simulation environment comprising 6,000 fault episodes; (4) an OpenTelemetry-Collector-inspired stress validation methodology; and (5) a comparative evaluation against static alerting, retry-based recovery, ungoverned automation, and MAPE-K without verification mechanisms.

2. BACKGROUND AND RELATED WORK

Research on self-healing systems originates from the field of autonomic computing [1], which introduced the concept of systems capable of monitoring, analyzing, planning, and executing corrective actions with minimal human intervention; the topic sits within the broader scope of applied computing and distributed-systems research [21]. The Monitor–Analyze–Plan–Execute over a shared Knowledge base (MAPE-K) architecture remains one of the most widely adopted frameworks for implementing adaptive behavior in distributed systems, and it has been formally modeled and analyzed for self-adaptation [5]. A substantial literature on self-adaptive software has developed landscape surveys [2], unifying reference models [3], surveys of autonomic degrees and models [4], run-time quantitative verification and sensitivity analysis [7], machine-learning-based adaptation [8], and consolidated research roadmaps [6]; self-healing software specifically has been surveyed more recently [20]. However, applying self-healing principles to observability pipelines presents additional challenges because recovery actions can directly affect the quality and availability of telemetry data used for operational decision-making.

Observability has become a fundamental requirement in cloud-native environments. Frameworks such as OpenTelemetry provide standardized mechanisms for collecting, processing, and exporting metrics, logs, and traces across heterogeneous infrastructures [16]. The OpenTelemetry Collector has emerged as a common telemetry aggregation layer due to its extensibility and vendor-neutral design [14], and guidance on its resiliency configuration documents the queueing, retry, and persistent-queue behaviors that this work later stresses [15]. Nevertheless, failures such as queue saturation, exporter timeouts, storage bottlenecks, backend outages, and configuration errors can disrupt telemetry delivery and reduce the reliability of monitoring and diagnostic processes.

Recent work in AIOps has focused on anomaly detection, root-cause analysis, fault localization, and automated incident response, including holistic frameworks to evaluate AI agents for autonomous clouds [9] and design principles for building such agents [10]. Surveys catalog root-cause-analysis methodologies, challenges, and trends in microservices [11], as well as anomaly detection and failure root-cause analysis in microservice-based cloud applications [12], and benchmarking frameworks such as RCAEval have been proposed to assess root-cause-analysis systems under realistic workloads and failure conditions [13]. While these studies contribute significantly to automated operations research, most assume that telemetry data remains sufficiently available and trustworthy throughout the diagnostic process.

The approach presented in this paper addresses a different but closely related problem: maintaining the reliability of the observability pipeline itself. Rather than focusing on application-level recovery, SHIELD-OP introduces a governance-oriented self-healing framework in which remediation actions are evaluated against policy constraints, operational risk thresholds, verification requirements, rollback capabilities, cooldown controls, and human-review conditions. This emphasis on governance and safety distinguishes the proposed framework from conventional feedback-loop automation and existing observability management approaches. It is complementary to work that assesses the readiness of telemetry itself for automated reasoning through multi-dimensional telemetry-quality scoring [22], and it echoes governance patterns—evidence review, risk scoring, and mandatory human approval—applied to multi-agent software-architecture decision-making [23]. Table 1 positions the research gap against these lines of work.

Table 1. Research gap and contribution positioning.

Area	Measures	Gap addressed
MAPE-K / self-adaptive systems	Feedback-loop structure	Limited observability-pipeline safety policy
OpenTelemetry Collector	Receive / process / export telemetry	Does not define remediation governance
AIOps / RCA benchmarks	Inject faults / export telemetry	Evaluate agents, not pipeline healing
Retry-only automation	Recover transient exports	Can worsen saturation / cardinality faults
SHIELD-OP (this work)	Policy-governed pipeline healing	Adds verification, rollback, risk budgets

3. PROBLEM DEFINITION

Consider an observability pipeline P that collects and processes telemetry data generated by a set of services S during a time interval W. The pipeline consists of components such as receivers, processors, buffers, exporters, storage backends, schema validation mechanisms, and operational control policies. A failure episode e is defined as any condition that degrades the availability, freshness, correctness, completeness, or usability of telemetry data flowing through the pipeline.

Given a set of remediation actions A, the objective is to select an action a ∈ A that restores normal pipeline operation while satisfying a predefined set of policy constraints C. These constraints may include limits on telemetry loss, restrictions on restart scope, protection of critical signal classes, cardinality-control requirements, rollback availability, and mandatory human review for high-risk actions. A remediation action is considered successful only when post-recovery verification confirms that the fault has been resolved and no policy violations have occurred.

The optimization objective extends beyond minimizing recovery time. Fast recovery achieved through excessive telemetry loss, disabled validation mechanisms, or unsafe operational actions may negatively affect downstream monitoring and diagnostic processes. Therefore, the problem is formulated as minimizing recovery time and telemetry degradation while maintaining a bounded probability of unsafe remediation actions. This formulation positions SHIELD-OP as a policy-constrained self-healing framework for observability infrastructure.

4. THE SHIELD-OP FRAMEWORK

SHIELD-OP contains seven components. The health monitor tracks queue depth, dropped spans, exporter success, ingestion delay, schema violations, memory pressure, and cardinality budgets. The fault classifier maps these observations to pipeline-fault classes. The policy governor checks risk, protected-signal rules, action budgets, and human-review thresholds. The remediation planner selects candidate actions. The executor applies bounded changes. The verifier checks whether recovery is real, using post-remediation probes analogous to Kubernetes liveness and readiness probes [17] and pod-lifecycle transitions [18]. The rollback manager restores the prior safe state when verification fails.

The core score used by the policy governor is a remediation risk score:

$$R(a, e) = b\_e + i\_a + u\_e - v\_a - r\_a \qquad (1)$$

where b_e is fault severity, i_a is estimated blast radius, u_e is uncertainty, v_a is verification strength, and r_a is rollback readiness. Actions above a threshold are routed to human review, consistent with risk-management guidance that recommends human oversight of consequential automated decisions [19]. Actions without verification probes are rejected for autonomous execution.

The policy store contains hard rules and soft budgets. Hard rules block actions that can remove protected telemetry classes, disable schema validation, or erase buffers without export confirmation. Soft budgets limit how frequently a collector may be restarted, how much sampling may be changed, how long a failover route may remain active, and how much telemetry loss is tolerated during recovery. Figure 1 shows the policy-governed self-healing loop and the components that participate in each cycle.

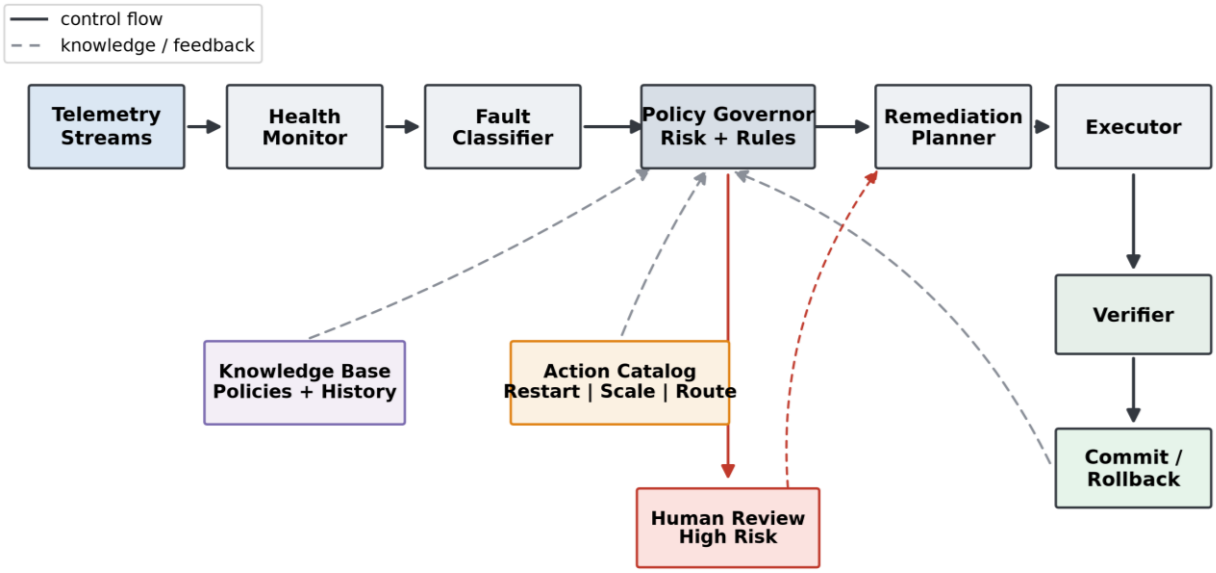

Fig 1. SHIELD-OP policy-governed self-healing loop for observability pipelines. Telemetry flows through a health monitor and fault classifier into a policy governor that scores risk against rules; admissible actions are planned, executed, and verified, with commit-or-rollback and human review of high-risk actions.

Table 2 summarizes the fault-to-action safety matrix: for each fault class it records a candidate remediation action, the verification condition that must hold for the action to be accepted, and the associated risk tier that determines whether human review is required.

Table 2. Fault-to-action safety matrix.

Fault	Candidate action	Verification condition	Risk
Queue saturation	Scale collector; reduce batch; route shard	Queue depth falling; no loss spike	Medium
Exporter outage	Failover exporter; enable buffer	Export success restored	Medium
Schema drift	Rollback schema; quarantine signal	Contract validation pass	High
Cardinality spike	Drop label; apply aggregation rule	Cardinality under budget	High
Memory pressure	Restart or scale; lower batch size	RSS and drop rate stable	Medium

5. METHODOLOGY

The evaluation of SHIELD-OP was conducted in two complementary stages. The first stage consisted of a reproducible simulation-based benchmark designed to compare alternative remediation strategies across a large set of observability-pipeline failure scenarios. The second stage involved an OpenTelemetry-Collector-inspired stress validation that examined the behavior of the framework under representative telemetry-pipeline failure conditions, including queue overflow, retry timeouts, collector crashes, persistent-queue behavior, configuration errors, and backend service unavailability, following the collector resiliency behaviors documented for production deployments [15]. This stress validation is intended to model common operational challenges observed in telemetry collection systems rather than to represent a production deployment.

The simulation benchmark used a fixed random seed schedule beginning with seed 20260608 and included 50 independent runs, each containing 120 failure episodes, resulting in a total of 6,000 evaluated fault scenarios. Six fault categories were considered: queue saturation, exporter outage, backend latency, schema drift, cardinality spikes, and memory pressure. SHIELD-OP was evaluated against four baseline approaches: static alert-based management, retry-only recovery, ungoverned automation, and a MAPE-K implementation without verification mechanisms. Performance was assessed using several operational metrics, including mean recovery time, 95th-percentile recovery time, telemetry-loss percentage, unsafe-action rate, human-review rate, and recovery-success rate. To support reproducibility, the random seed schedule, fault definitions, remediation policies, method configurations, and metric calculations are retained as supplementary artifacts, enabling independent replication of the experimental procedure without requiring access to production telemetry data.

Table 3. Synthetic evaluation setup.

Item	Configuration
Seeds	20260608 base seed; 50 independent runs
Episodes	120 failure episodes per seed; 6,000 total episodes
Fault classes	queue saturation, exporter outage, backend latency, schema drift, cardinality spike, memory pressure
Baselines	static alerts, retry-only, ungoverned automation, MAPE-K without verification
Metrics	mean MTTR, p95 MTTR, telemetry loss, unsafe-action rate, human-review rate, recovery success

6. ALGORITHM

Algorithm 1 presents the operational workflow of SHIELD-OP. The process begins by collecting pipeline health indicators over an observation window W. These indicators include metrics related to queue utilization, export success rate, ingestion latency, memory consumption, schema violations, and telemetry loss. The collected information is analyzed to identify the most probable fault category affecting the observability pipeline. Based on the detected fault, the

framework retrieves a set of candidate remediation actions from the action catalog. For each candidate action, a remediation risk score is computed using the policy-governed risk model of Section 4. Actions that violate mandatory constraints—including protected telemetry requirements, acceptable loss limits, or rollback availability conditions—are immediately discarded, and if the estimated risk exceeds a predefined threshold the incident is routed for human review rather than autonomous execution. Among the remaining admissible actions, the framework selects and executes the lowest-risk alternative, and a set of verification probes then evaluates whether the recovery objectives have been achieved before the outcome is committed or rolled back.

Algorithm 1. SHIELD-OP policy-governed self-healing workflow.

```
 1: Collect health indicators over window W
(queue, export success, latency, memory,
schema, loss)
 2: Analyze indicators; classify the most
probable fault category f
 3: Retrieve candidate remediation actions A_f
from the action catalog
 4: for each a in A_f:  R(a,e) ← b_e + i_a +
u_e − v_a − r_a
 5: Discard a violating hard rules (protected
telemetry, loss limit, rollback availability)
 6: if max admissible risk > τ:  route incident
to human review; return
 7: a* ← lowest-risk admissible action;
execute a*
 8: Run verification probes (queue depth,
exporter perf, ingestion delay, memory, drop
rate)
 9: if verification passes:  commit remediation
10: else:  roll back to prior verified state;
escalate for investigation
```

A distinguishing feature of SHIELD-OP is its verification-driven execution model. Unlike conventional automation approaches that treat action completion as success, SHIELD-OP requires measurable evidence of recovery before accepting a remediation outcome. Furthermore, all actions that modify routing policies, sampling configurations, schema controls, or buffering behavior must maintain rollback metadata prior to execution, ensuring safe recovery and repeatable operation under recurring fault conditions.

7. RESULTS

The results obtained from the Stage A evaluation are summarized in Table 4 across all 50 simulation runs. Among the automated approaches, SHIELD-OP achieved the lowest telemetry-loss percentage while maintaining a substantially lower unsafe-action rate than ungoverned automation. These findings indicate that the incorporation of policy constraints, verification procedures, and rollback mechanisms improves operational safety without significantly compromising recovery effectiveness.

Table 4. Stage A baseline comparison over 50 seeds.

Method	Mean MTTR	P95 MTTR	Loss %	Unsafe %	Success %
Static alerts	76.6	107.9	27.0	0.5	67.9
Retry-only	79.1	111.8	33.4	8.9	56.7
Ungoverned auto	44.5	65.6	20.8	29.5	64.7
MAPE-K no verify	52.0	75.4	21.1	16.7	71.0
SHIELD-OP	45.5	68.1	13.0	2.2	90.4

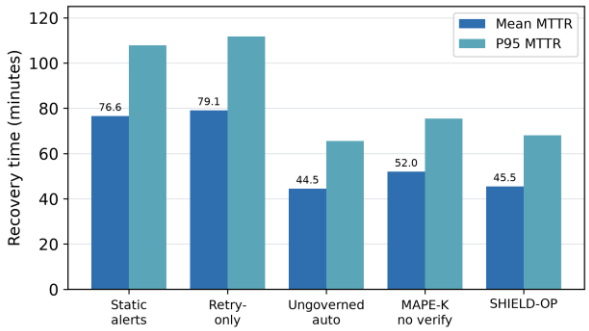

Fig 2. Mean and 95th-percentile recovery time across remediation methods (Stage A, 50 seeds). SHIELD-OP recovers nearly as fast as ungoverned automation while the human-gated baselines are far slower.

Static alert-based management exhibited the highest operational safety, with an unsafe-action rate of only 0.5%, because remediation decisions remained under human control; however, this approach also produced the longest recovery times (mean 76.6 minutes, p95 107.9 minutes) due to the delays associated with manual intervention. Retry-only recovery showed limited effectiveness across the evaluated fault classes: although retries were beneficial for transient exporter failures, they often provided little benefit for queue saturation, memory pressure, or persistent backend degradation, and produced the highest telemetry loss (33.4%). The MAPE-K baseline improved recovery performance compared with static alerting (mean MTTR 52.0 minutes) but generated a higher rate of unsafe remediation actions (16.7%) because recovery decisions were executed without explicit verification safeguards.

Figure 3 compares telemetry loss, unsafe-action rate, and recovery success across all methods on a common scale, and the separation is informative. SHIELD-OP recovers almost as quickly as ungoverned automation—45.5 versus 44.5 minutes mean MTTR, a difference of roughly 2%—yet cuts the unsafe-action rate from 29.5% to 2.2%, more than a thirteen-fold reduction, and lowers telemetry loss from 20.8% to 13.0%. It simultaneously attains the highest recovery-success rate of any method at 90.4%, some 19 percentage points above the next best (MAPE-K at 71.0%). Read together, these numbers answer RQ1 in the affirmative: policy-governed remediation retains almost all of the speed of unrestricted automation while removing the overwhelming majority of its unsafe actions, and it does so while improving rather than sacrificing recovery success.

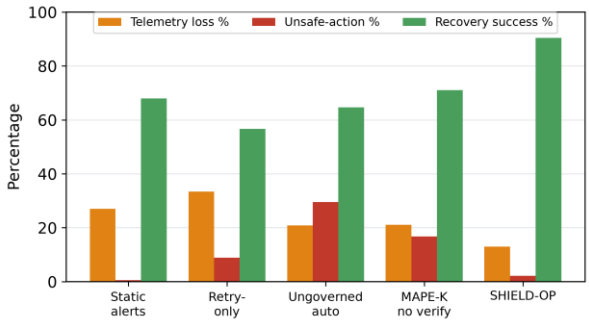

Fig 3. Telemetry loss, unsafe-action rate, and recovery-success rate across remediation methods (Stage A). SHIELD-OP combines the lowest loss and near-lowest unsafe rate with the highest recovery success.

8. COLLECTOR STRESS VALIDATION

The second stage of the evaluation focused on validating SHIELD-OP under failure conditions inspired by telemetry collector architectures. Unlike the synthetic benchmark used in Stage A, this validation emphasized operational behaviors commonly associated with observability pipelines, including in-memory queue overflow, retry timeouts, collector crashes, persistent-queue recovery, backend service unavailability, and configuration errors. The objective was to assess whether the proposed framework remains effective when confronted with failure scenarios that closely resemble challenges encountered in telemetry collection and delivery systems, and thereby to address RQ4.

The results of this evaluation are summarized in Table 5. Across the examined scenarios, SHIELD-OP consistently achieved lower telemetry-loss rates and fewer unsafe remediation actions than the comparison approaches. Default queue-and-retry mechanisms provided a basic level of resilience but were limited in their ability to address configuration-related faults and complex recovery situations. Persistent-queue support improved fault tolerance by reducing data loss during temporary disruptions; however, it did not address incorrect remediation decisions or operational policy violations. The MAPE-K baseline without verification demonstrated relatively fast recovery performance but exhibited a higher unsafe-action rate because recovery actions were executed without post-remediation validation.

Table 5. Stage B Collector-inspired stress validation.

Method	MTTR	Loss %	Unsafe %	Verified %
Default queue+retry	37.0	32.8	8.5	71.8
Persistent queue	31.4	25.0	5.7	77.0
MAPE-K no verify	28.9	23.7	14.2	75.2
SHIELD-OP	27.3	13.1	1.9	90.7

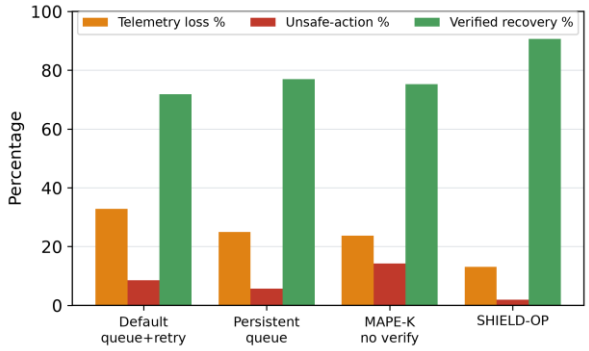

Fig 4. Telemetry loss, unsafe-action rate, and verified-recovery rate under Collector-inspired stress validation (Stage B). SHIELD-OP sustains the lowest loss and unsafe rate and the highest verified recovery.

The Stage B figures reinforce the Stage A pattern under more realistic collector dynamics. SHIELD-OP again recovers fastest in absolute terms (27.3 minutes) while holding telemetry loss to 13.1%—roughly half the 23.7–32.8% of the queue-based and MAPE-K baselines—and an unsafe-action rate of 1.9%, an order of magnitude below the 14.2% of the unverified MAPE-K variant. Its verified-recovery rate of 90.7% closely tracks its Stage A success rate, indicating that the verification-driven execution model transfers from the generic benchmark to collector-inspired conditions rather than being an artifact of the synthetic generator. By integrating policy enforcement, verification procedures, rollback capabilities, and human-review controls, SHIELD-OP maintains telemetry integrity while preserving efficient recovery behavior, which supports the claim that governance-oriented self-healing provides advantages beyond conventional resiliency mechanisms in observability environments.

9. ABLATION AND SAFETY ANALYSIS

To understand the contribution of individual framework components, an ablation study was conducted by selectively removing key elements of SHIELD-OP and measuring the resulting changes in performance. The results, summarized in Table 6, highlight the impact of policy governance, verification, rollback support, and risk budgeting on recovery effectiveness and operational safety.

Table 6. SHIELD-OP ablation results.

Configuration	MTTR	Loss %	Unsafe %	Review %
Full SHIELD-OP	45.5	13.0	2.2	43.8
No policy gate	37.3	14.3	10.1	24.1
No verifier	41.4	15.3	7.2	35.9
No rollback	51.0	17.6	4.7	38.6
No risk budget	40.1	13.9	6.1	27.6

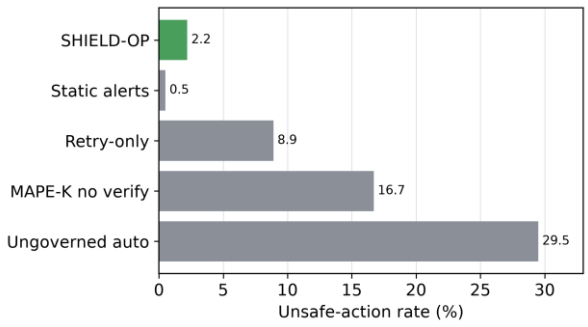

Fig 5. Unsafe-action rate across remediation methods (Stage A). Governance keeps SHIELD-OP near the human-gated static-alert baseline while the ungoverned and unverified methods rise sharply

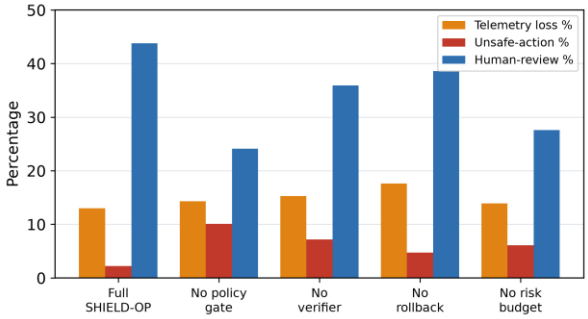

Fig 6. Ablation of SHIELD-OP governance mechanisms. Removing the policy gate, verifier, rollback, or risk budget each degrades a distinct safety or efficiency dimension

Removing the policy-governance layer produced a modest reduction in mean recovery time (45.5 to 37.3 minutes) because fewer remediation actions were delayed for review or policy evaluation; however, this improvement was accompanied by a substantial increase in unsafe remediation actions (2.2% to 10.1%, a more than four-fold rise) and a collapse in human-review coverage (43.8% to 24.1%), indicating that unrestricted automation introduces significant operational risk. Eliminating the verification component increased both telemetry loss (to 15.3%) and the unsafe-action rate (to 7.2%), as remediation decisions were accepted without confirming that the underlying fault had been resolved. Removing rollback capabilities produced the worst recovery time of any configuration (51.0 minutes) and the highest telemetry loss (17.6%), because without an automated mechanism for restoring a previously verified state, failed remediation attempts required additional corrective actions. Disabling risk-budget constraints reduced the frequency of human review (to 27.6%) but raised the unsafe-action rate to 6.1% in complex or uncertain fault scenarios.

Taken together, the ablation demonstrates that the effectiveness of SHIELD-OP depends on the interaction of multiple governance mechanisms rather than any single component. Policy enforcement most directly bounds unsafe actions, verification most directly protects telemetry integrity, rollback most directly protects recovery time, and risk budgeting most directly governs the human-review rate; each addresses a distinct source of operational risk. This decomposition answers RQ2 and RQ3: verification and rollback are together responsible for the framework's low telemetry loss and bounded recovery time, while the risk budget and policy gate are what route high-risk failures to human review rather than autonomous execution.

10. DISCUSSION

The findings of this study highlight the importance of treating observability pipelines as critical operational infrastructure rather than passive telemetry-transport mechanisms. Modern AI-assisted operations depend on the continuous availability of accurate metrics, logs, traces, and events for tasks such as anomaly detection, root-cause analysis, incident response, and automated remediation. When the observability pipeline itself becomes degraded, delayed, or unreliable, the effectiveness of these downstream operational processes is also compromised; the reliability of the telemetry substrate therefore bounds the quality of any automated diagnosis built on top of it [22]. Consequently, maintaining telemetry integrity should be considered an essential component of operational resilience.

A key observation from the evaluation is the trade-off between recovery speed and operational safety. Approaches based on ungoverned automation often achieve rapid recovery because actions are executed immediately and with minimal oversight; however, the experimental results demonstrate that this speed can come at the cost of a higher unsafe-action rate. SHIELD-OP adopts a different strategy by incorporating policy constraints, verification procedures, rollback mechanisms, and human-review controls for high-risk situations. Although these safeguards may introduce limited additional review overhead, they substantially reduce the likelihood of unsafe remediation decisions. This trade-off is particularly relevant in enterprise, regulated, and high-availability environments, where incorrect recovery actions can have consequences that exceed the cost of a slightly longer recovery process, and where governed, auditable automation is increasingly expected [19].

Another practical advantage of the proposed framework is its compatibility with existing observability ecosystems. SHIELD-OP does not require organizations to replace established telemetry collection, processing, or monitoring platforms. Instead, it can operate alongside existing observability infrastructure by consuming collector self-metrics, backend health indicators, and operational telemetry. Organizations may initially deploy the framework in an advisory mode that generates remediation recommendations without executing actions; as confidence in the governance policies and verification procedures increases, selected low-risk actions can gradually be automated. This phased adoption model reduces deployment risk while enabling organizations to progressively introduce self-healing capabilities into their observability environments.

11. THREATS TO VALIDITY AND FUTURE WORK

Several factors should be considered when interpreting the results of this study. The primary limitation is that the evaluation was conducted in a controlled simulation environment rather than a production deployment. Although the collector-oriented stress validation incorporates failure scenarios commonly associated with observability infrastructures, it remains an abstraction of real-world operational environments. Consequently, the reported results should be interpreted as evidence of the framework's behavior under the defined experimental conditions rather than as direct indicators of production performance.

Another potential limitation is the scope of the evaluated fault classes. While the selected scenarios cover a range of common observability-pipeline failures, real-world deployments may experience more complex interactions involving heterogeneous infrastructure, dynamic workloads, multi-tenant environments, and organization-specific operational policies. In addition, the

remediation effectiveness observed in the study may be influenced by the assumptions embedded within the simulation model and policy configurations.

Future research should focus on validating SHIELD-OP in operational testbeds built around OpenTelemetry Collector deployments and cloud-native observability platforms. Evaluations using publicly available AIOps benchmarks and fault-injection frameworks, such as holistic autonomous-cloud evaluation environments [9] and root-cause-analysis benchmarks [13], would provide additional evidence of the framework's effectiveness under realistic workload conditions and would allow assessment of whether observability-pipeline self-healing improves the quality of evidence available to automated diagnostic systems. Further work may also explore adaptive policy management techniques that learn appropriate risk thresholds from historical incidents and operational feedback while preserving non-negotiable safety constraints. From a deployment perspective, production-grade implementations would benefit from tighter integration with orchestration platforms, telemetry collectors, configuration-management systems, routing services, and incident-management workflows.

12. CONCLUSION

This paper presented SHIELD-OP, a policy-governed self-healing framework for observability pipelines in distributed cloud environments. Unlike traditional self-healing approaches that focus primarily on application services, SHIELD-OP addresses the resilience and operational safety of the telemetry infrastructure itself. The framework integrates continuous monitoring, fault classification, policy-aware decision-making, remediation planning, verification mechanisms, rollback procedures, and human-review controls to support reliable recovery from observability-pipeline failures.

The proposed approach was evaluated through a two-stage experimental methodology consisting of a large-scale simulation benchmark and a collector-oriented stress validation. The results demonstrated that SHIELD-OP achieves lower recovery times and reduced telemetry loss than static alerting and retry-based recovery approaches while maintaining a substantially lower unsafe-action rate than ungoverned automation. The ablation analysis further showed that policy governance, verification, rollback support, and risk-aware controls each contribute to the framework's overall effectiveness and safety. The primary contribution of this work is a governance-oriented remediation architecture for observability pipelines. By improving the reliability and integrity of telemetry data, SHIELD-OP helps ensure that AI-assisted monitoring, diagnostic, and self-healing systems can operate using more dependable operational evidence, providing a foundation for future research on safe and trustworthy automation within cloud-native observability ecosystems.

13. ACKNOWLEDGMENTS AND AI DISCLOSURE

The author thanks the broader research and engineering communities working on distributed systems, observability, OpenTelemetry, AIOps, self-adaptive systems, and software reliability. Generative AI tools were used to assist with outline refinement, drafting support, and editorial review. The author verified all technical claims, simulation setup, figures, references, and conclusions and remains fully responsible for the content of this manuscript.

14. REFERENCES

[1] Kephart, J. O., and Chess, D. M. 2003. The Vision of Autonomic Computing. Computer 36, 1, 41–50.

[2] Salehie, M., and Tahvildari, L. 2009. Self-Adaptive Software: Landscape and Research Challenges. ACM Transactions on Autonomous and Adaptive Systems 4, 2.

[3] Weyns, D., Malek, S., and Andersson, J. 2012. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems. ACM Transactions on Autonomous and Adaptive Systems 7, 1.

[4] Huebscher, M. C., and McCann, J. A. 2008. A Survey of Autonomic Computing: Degrees, Models, and Applications. ACM Computing Surveys 40, 3.

[5] Arcaini, P., Riccobene, E., and Scandurra, P. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In IEEE/ACM SEAMS.

[6] de Lemos, R., et al. 2013. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. Lecture Notes in Computer Science, vol. 7475, Springer.

[7] Filieri, A., Tamburrelli, G., and Ghezzi, C. 2016. Supporting Self-Adaptation via Quantitative Verification and Sensitivity Analysis at Run Time. IEEE Transactions on Software Engineering.

[8] Gheibi, O., Weyns, D., and Quin, F. 2021. Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review. ACM Transactions on Autonomous and Adaptive Systems.

[9] Chen, Y., et al. 2025. AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds. arXiv:2501.06706.

[10] Shetty, M., et al. 2024. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. In ACM Symposium on Cloud Computing.

[11] Wang, T., and Qi, G. 2024. A Comprehensive Survey on Root Cause Analysis in Microservices: Methodologies, Challenges, and Trends. arXiv:2408.00803.

[12] Soldani, J., and Brogi, A. 2022. Anomaly Detection and Failure Root Cause Analysis in Microservice-Based Cloud Applications: A Survey. ACM Computing Surveys.

[13] Pham, L., Ha, H., and Zhang, H. 2025. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems. In Proceedings of the ACM Web Conference.

[14] OpenTelemetry. 2026. OpenTelemetry Collector. Documentation.

[15] OpenTelemetry. 2026. Collector Resiliency. Documentation.

[16] Cloud Native Computing Foundation. 2025. OpenTelemetry Specification.

[17] Kubernetes. 2026. Configure Liveness, Readiness and Startup Probes. Documentation.

[18] Kubernetes. 2026. Pod Lifecycle. Documentation.

[19] National Institute of Standards and Technology. 2023. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1.

[20] Yazdanparast, Z. 2024. A Survey on Self-Healing Software Systems. arXiv:2403.00455.

[21] International Journal of Computer Applications. 2026. IJCA Scope and Topics. Foundation of Computer Science.

[22] Gullapalli, S. 2026. Telemetry Quality Indexing for AI-Ready Observability in Multi-Node Cloud Systems: A Multi-Dimensional Framework for Distributed Infrastructure Diagnosis. International Journal of Engineering Development and Research (IJEDR) 14, 2, 878–. DOI: 10.56975/ijedr.v14i2.308419.

[23] Divi, V. R. 2026. Multi-Agent Debate for Software Architecture Governance: A Framework for ADR Generation, Risk Review, and Deployment Readiness. International Journal of Engineering Development and Research (IJEDR) 14, 2, 862–. DOI: 10.56975/ijedr.v14i2.308420.